813-RD-009-001

EOSDIS Core System Project

DADS Prototype Two Multi-FSMS Product Integration Evaluation

November 1994

Hughes Applied Information Systems
Landover, Maryland

DADS Prototype Two Multi-FSMS Product Integration Evaluation for the ECS Project

November 1994

Prepared Under Contract NAS5-60000

APPROVED BY

Steve Fox /s/	11/28/94
Steve Fox, SDPS Office Manager EOSDIS Core System Project	Date

Hughes Applied Information Systems
Landover, Maryland

813-RD-009-001

This page intentionally left blank.

Preface

This document contains the prototype results report for DADS Prototype Two. This document is submitted as required by the ECS Statement of Work, Section 3.3.3.3, and does not require Government approval. The results of this prototype will also be documented in the Prototyping and Studies Final Report (DID 333/DV3).

For additional technical information pertaining to this DADS Prototype, contact Tom Smith or Chris Holmes, SDPS-DADS, at (301) 925-0647/(301) 925-0657 or on email at tsmith@eos.hitc.com / cholmes@eos.hitc.com.

Questions concerning the control or distribution of this document should be addressed to:

Data Management Office
The ECS Project Office
Hughes Applied Information Systems
1616 McCormick Dr.
Landover, MD 20785

This page intentionally left blank.

Contents

Figures

Tables

This page intentionally left blank.

1. Introduction

1.1 Identification

This Data Archive and Distribution System (DADS) Prototype Two Results Report is prepared for the Earth Observing System Data and Information Systems (EOSDIS) Core System (ECS) project, contract number NAS5-60000.

1.2 Summary

Prototype 2 has confirmed that multiple heterogeneous File Storage Management Systems (FSMS) products can be used as components in a managed distributed data storage environment. File requests can be routed to the appropriate FSMS via the use of a Universal Identifier (UID). Router failures do not affect requests already in the appropriate FSMS request queue. Multiple routers can be used to minimize the risk of total failure.

1.3 Purpose

The primary purpose of Prototype 2 is to functionally evaluate data retrievals in a heterogeneous storage environment consisting of two or more storage product strings. The term string denotes a complete system configuration consisting of a host, a data management product, and a tertiary storage device. Two FSMS products were selected for use in this prototype: the Archival Management and Storage System (AMASS) product from Advanced Archival Products and the EpochServ product from Epoch Systems, an EMC^2 Company.

The AMASS product consists of a separate file system that is closely connected to the UNIX File System (UFS) via the Virtual File System (VFS) interface at the Unix kernel level. The EpochServ product resides entirely within the UFS.

1.4 Approach

The prototype approach involves a three step process:

1. Store an appropriate amount of data into each FSMS product to facilitate testing. Use this opportunity to measure the native storage and retrieval capabilities of each FSMS product in the prototype.
2. Test each product using commonly available access methods in Unix systems. These tests included: Network File System (NFS), Remote Copy (RCP), and File Transfer Protocol (FTP).
3. Create a Location Server/Request Router (LSR) software product to handle requests to multiple FSMS products. Evaluate the viability and impacts of this approach.

1.5 Parent Document

February 1993	ECS Statement of Work

1.6 Applicable Documents

The following documents are applicable to this document:

193-707-PP1-002	ECS Prototype Results Review, submitted December 1993
193-216-SE1-001	ECS Requirements Specification, submitted February 1994
193-317-DV1-001	ECS Prototyping and Studies Plan, submitted May 1993
193-318-DV3-005	ECS Prototype and Studies Progress Report, submitted November 1993

2. Hardware Configuration

The following hardware configuration was used for Prototype 2.

AMASS HOST:	HP 9000/800 E35
OS VERSION:	HP-UX A.09.04
DISK SPACE:	4 GB - AMASS Disk Cache 685 MB
ROBOTIC UNIT:	Metrum RSS-48
CARTRIDGE FORMAT:	19 mm SVHS - Helical Scan Magnetic Tape
CARTRIDGE CAPACITY:	48 Cartridges
VOLUME CAPACITY:	18 GB/Cartridge (approximate)
TAPE TRANSPORTS:	2
DATA TRANSFER INTERFACE:	SCSI-1

EPOCH HOST:	SUN SPARC II
OS VERSION:	SunOS 4.1.3_U1
DISK SPACE:	2.3GB
	EpochServ Software & Directory Space 1.1 GB
EPOCH CLIENT:	Sun IPX
OS VERSION:	SunOS 4.1.3_U1
DISK SPACE:	933 MB
	Epoch Client Software & Directory Space 189 MB
ROBOTIC UNIT:	HP 20C
CARTRIDGE FORMAT:	5.25 in - Magneto Optical Disks
CARTRIDGE CAPACITY:	32 Cartridges
VOLUME CAPACITY:	650 MB/Cartridge
TAPE TRANSPORTS:	2
DATA TRANSFER INTERFACE:	SCSI-1

LSR HOST:	SGI Challenge XL (8 CPU)
OS VERSION:	IRIX 5.2
DISK SPACE:	20 GB

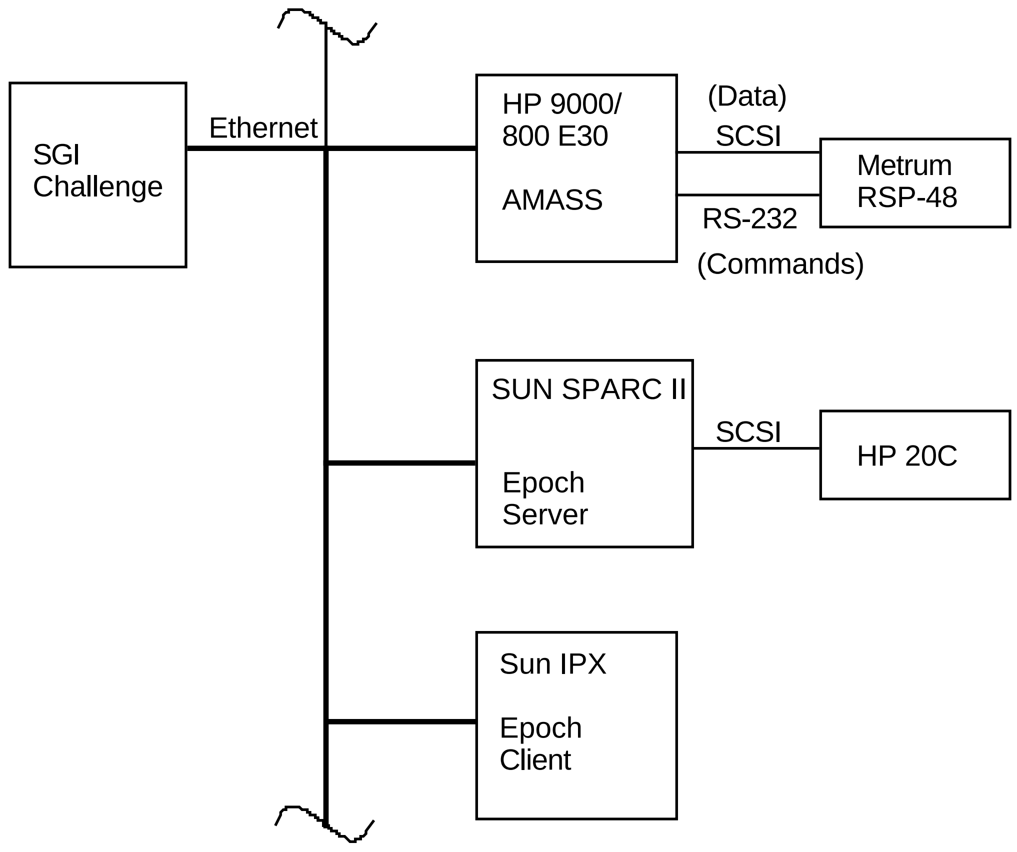

Figure 2-1. Prototype 2 Organization

3. AMASS 4.2.1

3.1 Product Summary

AMASS is a separate file system product that is attached to the UNIX Operating System (OS) at the VFS of the OS kernel (See Figure 3-1). From the perspective of the OS, AMASS is another UFS-like File System. All functions that work in the UFS will work in AMASS and all network communications protocols available to the OS are available to AMASS. The product appears as a endless UFS that is only limited by the available tertiary storage.

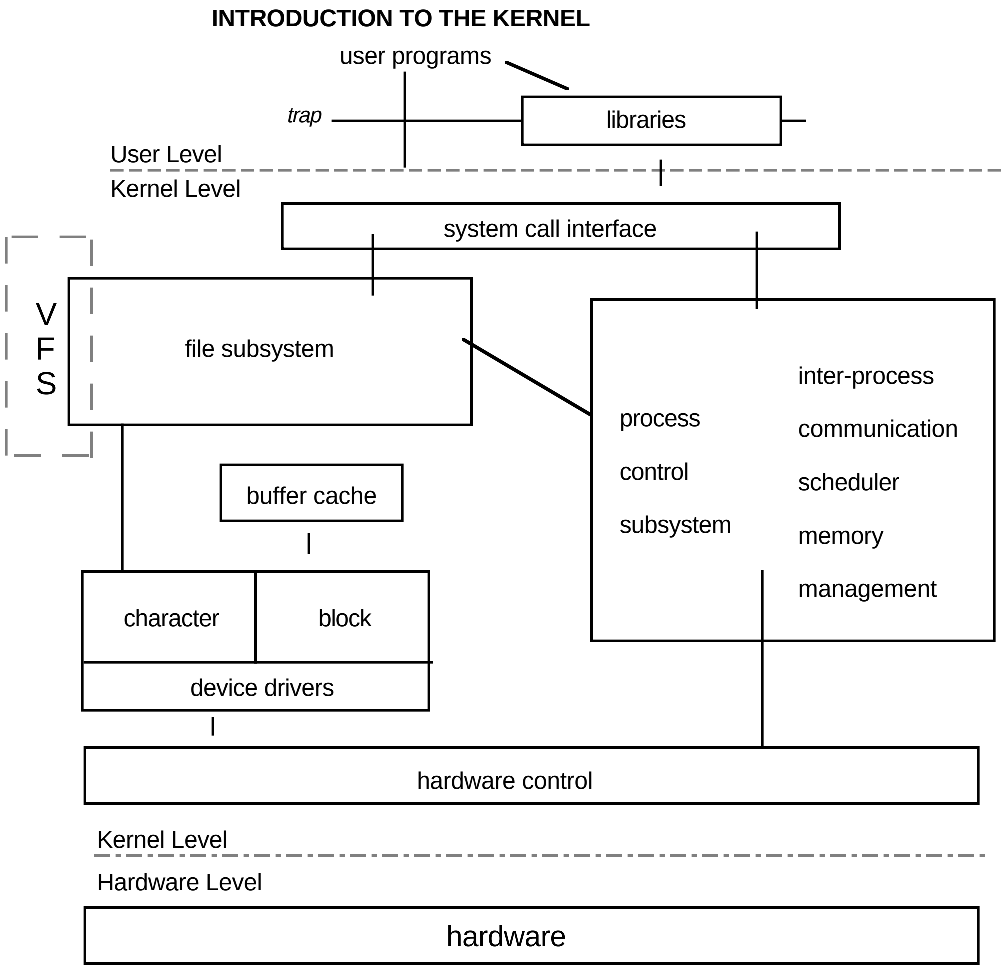

Figure 3-1. UNIX OS Kernel & VFS

AMASS is hierarchical in nature, but it is not a hierarchical storage system in the same way as most Unix resident products. The directory structures for the AMASS file system are kept in a RAIMA database (at this point, the formats for RAIMA are proprietary,) on disk (within UFS) with periodic backups to tertiary storage. A transaction log, listing transactions since the last backup, is also kept on disk to facilitate rapid file recovery in the event of system failure. There is no Unix Index node (Inode) for each file managed by AMASS. The RAIMA database keeps track of each file's Inode information as well as its location on tertiary storage. This data and other associated per file data, forms the AMASS Master File Directory (MFD). AMASS files reside in a separate file system that interfaces to the UFS via the VFS layer. This layer allows the AMASS file system to appear as a UNIX File System to users. Each Unix command is interpreted by the AMASS software and a query is generated against the RAIMA database. This database provides the necessary Unix Inode or file retrieval information to satisfy the query (e.g., *ls, ls -l, cat, etc*.). Though Unix commands are used on specific data files, these files actually reside in the AMASS cache area unless the file is specifically copied into the UFS.

A disk area is set aside exclusively to cache AMASS data. Data files are copied directly to tertiary media either when the write operations to the cache are completed or in parallel. Thus, there is no migration in the traditional sense. Data retrieval is on a block or file basis with data staging on a demand basis. Unix directories can be associated directly to specific storage devices or specific storage volumes via links in the RAIMA database. This system of database links via physical Unix Inodes has several advantages when dealing with tertiary storage. Files may be easily moved between directories, and the file system is not limited by the Inodes available to the host file system.

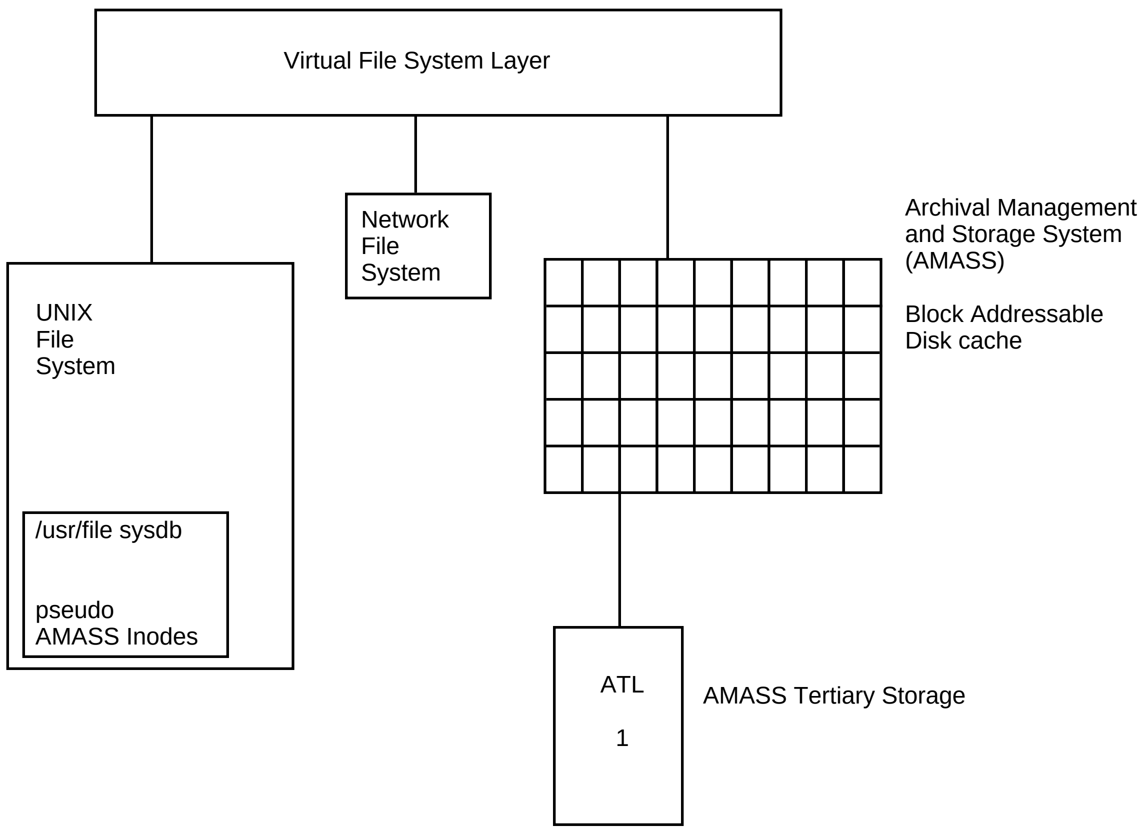
Virtual File System Layer
Network File System
UNIX File System
/usr/file sysdb
pseudo AMASS Inodes
Archival Management and Storage System (AMASS)
Block Addressable Disk cache
ATL
1
AMASS Tertiary Storage

Figure 3-2. AMASS Organization

This page intentionally left blank.

4. EpochServ 6.0.4

Epoch entered the mass storage market in 1989 with the Epoch-1 Infinite Storage Server. This turnkey storage system consists of both server and storage hardware. The advent of Open Systems drove the development of the Epoch-2 Data Server which is a fully integrated SPARC-based network data server. A recent change in corporate philosophy has resulted in the EpochServ product. This is a full function product based on the Epoch-2 software but it is Epoch hardware independent.

4.1 Product Summary

EpochServ provides virtually infinite magnetic disk free space by transparently moving inactive files to tertiary storage. EpochServ migrates files automatically to tertiary storage based on system administrator defined criteria. Staging is on a demand basis and is transparent to the user. The product can function as a system server or as a network server.[1] Epoch is following Open System Standards. They are currently redesigning their software to remove required OS kernel modifications. This will streamline the product port process. Epoch currently offers the following products:

Epoch Library Manager

Part of EpochServ. Controls allocation, scheduling, and tracking of all removable media for tertiary optical disk and tape library units.

Epoch Migration Manager

Part of EpochServ. Manages the space on an EpochServ system's magnetic disks and provides file migration among various levels in the storage hierarchy.

Epoch Backup Manager

Part of EpochServ. Provides automatic, unattended, on-line backups of the server's files in the storage hierarchy.

Epoch Migration Manager/MLS

A layered Epoch Migration Manager product that provides Multi-Level Staging (MLS) of files from one type of tertiary storage to another.

Epoch Backup Manager/DR

A layered Epoch Backup Manager product that provides disaster recovery (DR) services for the Epoch Backup Manager.

[1]Separate client code modules are required for this function.

Epoch Migration/Client

A layered product that places client code on other network systems to allow file migration from individual workstations and servers to and from an EpochServ central server. Provides client machines with the appearance of an unlimited virtual disk system.

Epoch Backup/Client

A layered product that places client code on other network systems to extend EpochServ's backup capabilities to individual workstations and servers across the network.

4.2 Epoch Configuration

EpochServ resides within the UFS. All Unix system commands are available to the EpochServ user. Epoch also provides Epoch Specific extensions of UFS commands designed to work specifically with the mass storage system (e.g., *epls, epmount*, *etc.*). EpochServ can be accessed remotely via all UFS supported protocols or directly using the native file system and either the UFS command set or enhanced EpochServ commands. EpochServ does not modify existing fields in the Unix Inode. An unused field in the Inode is used to point to a separate flat file of attributes kept for all files being managed by EpochServ. This file is consulted for file access, file migration, and backup and restore. The current Epoch system uses a concept called 'wrappers' that intercept commands that appear to be FSMS related at the kernel level. This approach is cumbersome, difficult to port, and version dependent. EpochServ 6.1 will correct this problem by making modifications to the NFS Daemon.

Each access, movement, or modification to a stored data file is recorded in log files, and the appropriate indexes are updated in the attribute file. EpochServ has the concept of "Staging Trails" which is used to group data from a particular file system onto specifically designated physical volumes or devices. Files can be assigned to a staging trail based on an algorithm or via directory structure associations. The software limits the maximum number of staging trails supported to 100. Performance begins to degrade at numbers above 20. The product does not support a Data Class/Family of Files concept. EpochServ uses the Mass Storage Reference Model (MSRM) as one of the factors in its design. The equivalent of an MFD exists and can be examined and edited by the system administrator. EpochServ does not provide external naming services for bitfiles.

Each volume in EpochServ is electronically labeled by the system. This label is always checked prior to any Read/Write (R/W) operation. EpochServ transparently moves files from on-line to tertiary storage and vice versa. EpochServ uses three storage watermarks for efficient storage utilization: Pre-Stage Water Mark (PSWM), Low Water Mark (LWM), and High Watermark (HWM). EpochServ creates a candidate list of files that can be migrated. Candidate files are pre-staged to tertiary storage but their image remains on magnetic disk in the PSWM area. When utilization reaches the HWM, files in the PSWM are deleted from on-line storage.

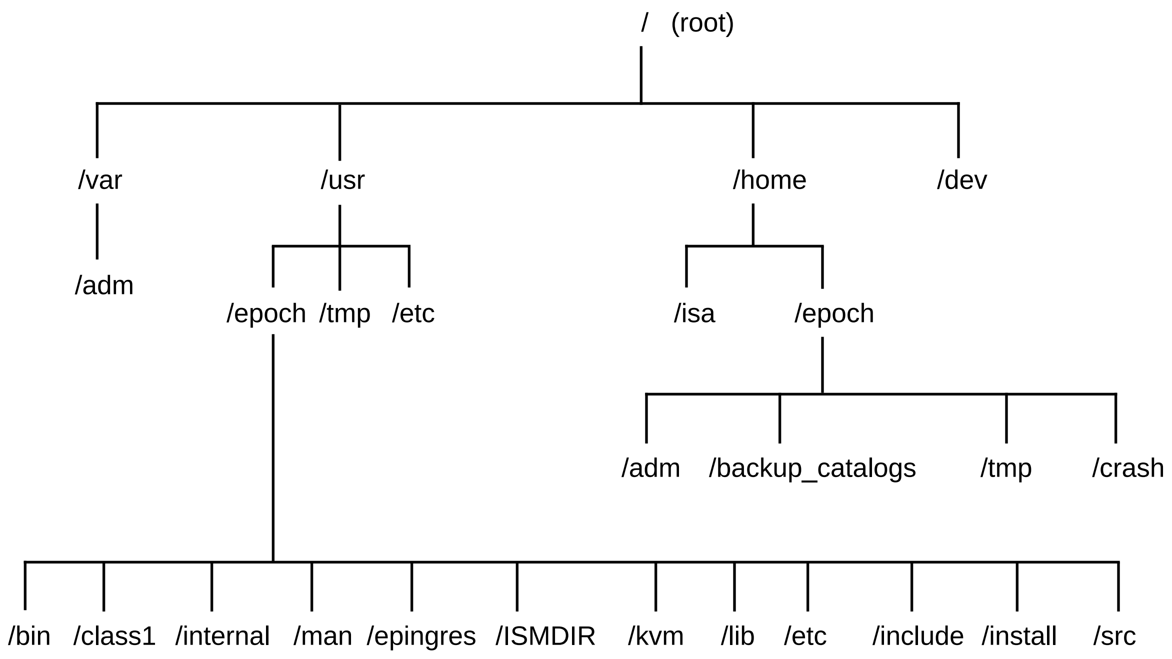

Figure 4-1. Epoch Organization

This page intentionally left blank.

5. Prototype Implementation

The primary goal for the prototype was to prove that multiple FSMS products in a heterogeneous environment could be accessed independently without data or request loss in a timely manner. The method selected to prove this concept was the development of an LSR, which provided a single point of access to successfully route requests for data to the correct FSMS in the heterogeneous architecture. Each FSMS needed to receive and process multiple simultaneous requests, retrieve the files from their data stores, and transfer the resultant files to the host specified in the request. The LSR itself could not be a single point of failure; if the LSR went down for some reason, processing for data already requested must be able to continue and another LSR process (possibly on a different machine) must be able to continue where the last one left off. For this prototype only one LSR host was used. This was an Silicon Graphics, Inc. (SGI) Challenge (IRIX System V.5.2) running the custom LSR code. Two FSMS hosts were connected via sockets to the LSR.

5.1 Prototype Design

The prototype design incorporated the primary goal of FSMS heterogeneity as well as the secondary goal of not creating a single point of failure in the file retrieval and request routing mechanisms. The tertiary goal was to minimize custom software complexity by placing file retrieval management responsibilities on the individual FSMS products. The prototype design allowed each FSMS to run simultaneous requests and allowed each one to be connected to multiple LSRs. A universal file identifier was used so that the LSR could easily determine which FSMS was serving a file.

The FSMS products selected for this prototype included the AMASS product and an Epoch Migration Client as well as an Epoch Host. The epoch client added an additional level of staging and migration which was used to better examine a heterogeneous distributed storage environment.

5.2 FSMS Server Design

The FSMS Server code ran on each machine that hosted an FSMS. The main process would open a socket on a well known endpoint and listen for requests for connection. This process is the FSMS connection server. When a request for a connection came in, the main process executed a fork(). The parent process closed its connection to the requester and again listened for more connection requests. The child process was then allowed to interact with the requester to service requests. This process is the FSMS server for a particular LSR. By keeping an FSMS connection server up all the time, it allowed multiple LSRs to connect to the server. If an LSR went down, another LSR could connect (or possibly already be connected) and software above the LSR could re-route the requests to one that was operational.

The process servicing the request would first tell the host what type of FSMS it was and what UID stubs that it was serving. The host would then tell the FSMS its *hostname*. After this processing, the FSMS simply waited for file requests from its LSR. When a request for a file came, the FSMS again executed a fork(). The parent process went back to waiting for more requests from its LSR and the child process retrieved the file requested and transferred it, via FTP, to the host specified. This allowed multiple simultaneous requests that would continue if the LSR went down.

5.3 LSR Server Design

The LSR Server used a host setup file that listed all of the possible FSMS Server hosts. It read each server entry and attempted to connect to the well known endpoint. If the connection was successful, it asked the FSMS server what type of FSMS and which UID it was serving. It also told the FSMS its hostname so that the FSMS knew where to send the files. This information was put in a table for use by the router. The entire host setup file was read until each host was tried and all of the hosts that responded were put in the router table.

When a request came in (in this case, was entered by the keyboard), the router table was checked to see if the UID specified was being served by any of the available FSMSs. If it was, the request was sent over the socket to the appropriate FSMS server. If not, an error was posted to the log and the user (in this case, the terminal). The LSR Server would not wait for a response from the FSMS server, but would continue processing subsequent requests.

5.3.1 LSR Responses

Responses to the LSR were handled by a separate process that simply polled a well known directory (in this case, /tmp/tool) for files that fit the UID scheme. When it found a file, it put an entry in the log and deleted the file. This was considered a response from the FSMS server. In order to not risk hanging an FTP session, the FTP process on the FSMS server would transfer the file under a non-UID name and at the end of the transfer, rename the file to its original name. This way, no partially transferred files would be deleted.

5.4 Problems with the Prototype

Several problems inherent in the prototype design were found, mostly focused on the fork() call and how it interacted with waiting processes.

First, when a process does a fork() call, it associated the child process with the parent process. In the first version of the tool, when a child process attempted to die, via the exit() call, it would almost invariable do it when the parent was waiting on a socket for another request. The OS would not allow the child to die without notifying the parent, so the process table started filling with defunct processes. After a while, the host machine could no longer create any processes, and the FSMS server was basically dead. To alleviate this condition, the signal() command was used so that the parent process would ignore the SIG_CHLD signal. On the Hewlett Packard (HP), all of the defunct processes disappeared and there was no longer a problem of running out of processes, but this did not help under the SunOS. Other attempts were made to clean up this problem on the Sun, but the problem would not go away completely. The defunct processes did die after a while and the server never went down due to the limited number of processes (actually, it never went down at all), so the problem was ignored for now.

The second problem was a little harder to quantify, and there was not sufficient time to fully investigate. It seemed that some of the child processes would not continue processing if the parent process was waiting for something. Not all of the child processes in a batch would do this. This behavior was observed on both platforms. For testing, five requests would be sent to each FSMS connected and the requester would wait for a period of time before sending more, allowing the FSMS to service the requests and be ready for more. But, on the HP, usually three of the processes would complete and the other two would wait until more requests came in. It did not seem to matter how long the requester waited, the last two were always slow. It also seemed that when the main processes were killed (for a restart), all of the child processes would finish very quickly.

5.5 Testing Procedure

Files were created in several sizes, each in a common pattern. The first longword of each file contained the file size and every longword after that contained a long integer counting up from 2 to the size of the file. This allowed very easy checking of files coming out of the archives for validity.

Each file was created on a staging area and then copied into the disk area controlled by the FSMS on that system. For the Epoch systems, an `epstage` command was issued to stage the file to the optical disk and this command was timed, and this time was posted in the data (representing a system response for a file size). Due to the nature of AMASS, the Unix copy (`cp`) was timed. Also, for AMASS, since there is so much caching, we had to run a much larger amount of data through to counteract the cache. With enough data running through to bog down the cache, we could get a better idea of a system throughput. Each store request was synchronous and there were no other users on the system.

Each file was put into a specific directory, each directory having a keyword associated with it to facilitate a UID. Three (Epoch) or four (AMASS) directory levels were used to test the response time for different transfer mechanisms according to how far down in a directory tree they were. The directories were maintained by file size; there was a root directory which had subdirectories for all the level one files, and each of these subdirectories were the root for the level two files of the same size and so on.

The retrieve times for native, RCP, NFS, and FTP were also tested synchronously. Each request for a file (on Epoch, a Epoch Bulk Stage In (`epbsi`) was issued; on AMASS, a Unix `cp` was issued) was the only request active on the FSMS and each request was timed. This allowed us to baseline the performance of each approach and compare them against one another. Unfortunately, this number does not reflect a quiescent network. The prototype equipment was placed on the same network segment as the majority of the Landover equipment. This created some significant disparities between data readings. This is particularly true of individual data points used for the averages (See Appendix A) and of network file movement (i.e., FTP, NFS, RCP).

Table 5-1. AMASS - Data Distribution (1 of 2)

	Directory Level 1		Directory Level 2		Directory Level 3		Directory Level 4	
	files	avg time (sec)	files	avg time (sec)	files	avg time (sec)	files	avg time (sec)
64K								
Number Archived:	15000		1000		1000		1000	
Number Retrieved:								
Native:	2000	0.55	1000	0.46	1000	0.97	1000	0.38
FTP:	1000	1.41	500	1.25	500	1.62	500	1.23
NFS:	1000	2.22	500	0.47	500	0.95	500	0.48
RCP:	1000	1.51	500	1.45	500	1.85	500	1.47
100K								
Number Archived:	15000		1000		1000		1000	
Number Retrieved:								
Native:	1500	0.72	1000	1.83	1000	0.58	1000	0.59
FTP:	0	0	0	0	0	0	0	0
NFS:	1000	4.46	500	1.97	250	0.69	250	0.67
RCP:	1000	2.17	500	2.82	250	1.57	250	1.53
500K								
Number Archived:	5000		250		500		500	
Number Retrieved:								
Native:	1000	1.68	150	1.31	500	1.49	500	1.57
FTP:	1000	2.41	50	2.27	50	9.17	50	9.10
NFS:	1000	2.38	50	2.49	50	7.97	50	7.73
RCP:	1000	2.73	50	2.72	50	10.04	50	10.55
1M								
Number Archived:	3500		250		250		250	
Number Retrieved:								
Native:	800	3.17	250	29.56	250	2.69	250	2.73
FTP:	1000	3.55	50	28.84	50	3.71	50	4.43
NFS:	1000	3.34	50	28.35	50	4.29	50	4.22
RCP:	1000	3.88	50	29.71	50	3.95	50	4.05

Table 5-1. AMASS - Data Distribution (2 of 2)

	Directory Level 1		Directory Level 2		Directory Level 3		Directory Level 4	
	files	avg time (sec)	files	avg time (sec)	files	avg time (sec)	files	avg time (sec)
50M								
Number Archived:	200		20		10		10	
Number Retrieved:								
Native:	250	88.82	10	142.28	10	149.05	10	146.74
FTP:	25	183.88	5	165.73	5	165.46	5	160.57
NFS:	25	218.83	5	212.51	5	202.46	5	201.11
RCP:	25	200.84	5	174.74	5	169.18	5	161.05
100M								
Number Archived:	100		10		10		10	
Number Retrieved:								
Native:	300	177.73	10	239.68	10	234.05	10	241.00
FTP:	50	294.38	5	334.54	5	349.06	5	323.47
NFS:	25	381.33	5	348.63	5	339.96	5	330.81
RCP:	50	289.01	5	330.72	5	323.91	5	327.39
500M								
Number Archived:	10		10		10		10	
Number Retrieved:								
Native:	10	953.61	6	850.07	10	940.61	10	964.46
FTP:	6	1281.52	5	1216.52	5	1243.97	4	1213.26
NFS:	5	1526.14	5	1479.61	5	1486.69	4	1532.08
RCP:	5	1488.20	5	1446.53	5	1422.46	5	1365.44
1G								
Number Archived:	10		10		10		10	
Number Retrieved:								
Native:	13	1884.99	10	1885.86	10	1865.74	10	1881.16
FTP:	5	2399.62	3	2373.15	3	2333.68	3	2582.84
NFS:	5	3317.59	3	3556.15	3	2920.43	3	3049.20
RCP:	5	2561.73	3	2529.45	3	2502.62	3	2644.24
Total Files 46,950								

Table 5-2. Epoch Host - Data Distribution (1 of 2)

	Directory Level 1		Directory Level 2		Directory Level 3	
	files	avg time (sec)	files	avg time (sec)	files	avg time (sec)
64K						
Number Archived:	1000		1000		1000	
Number Retrieved:						
Native:	1000	0.66	900	0.84	900	0.89
FTP:	0	0	0	0	0	0
NFS:	1000	0.36	1000	0.44	1000	0.44
RCP:	0	0	50	2.17	50	1.89
100K						
Number Archived:	1000		500		500	
Number Retrieved:						
Native:	1000	0.93	500	1.27	500	1.07
FTP:	0	0	0	0	0	0
NFS:	1000	0.67	500	0.68	500	0.69
RCP:	100	2.03	50	2.03	50	2.33
500K						
Number Archived:	500		200		50	
Number Retrieved:						
Native:	500	2.46	200	1.81	50	2.44
FTP:	100	4.84	100	4.34	25	4.80
NFS:	500	1.98	200	2.04	50	2.09
RCP:	100	3.17	25	3.28	25	3.31
1M						
Number Archived:	50				20	
Number Retrieved:						
Native:	50	4.18	20	3.51	20	4.17
FTP:	25	7.75	10	8.66	10	7.74
NFS:	50	3.25	20	3.89	20	3.52
RCP:	25	4.43	10	7.34	10	5.46

Table 5-2. Epoch Host - Data Distribution (2 of 2)

	Directory Level 1		Directory Level 2		Directory Level 3	
	files	avg time (sec)	files	avg time (sec)	files	avg time (sec)
50M						
Number Archived:	10		5		5	
Number Retrieved:						
Native:	10	122.43	5	104.47	5	131.73
FTP:	5	275.49	3	259.91	3	294.58
NFS:	10	121.54	5	120.99	5	123.70
RCP:	5	112.58	5	109.44	5	107.73
100M						
Number Archived:	10		5		5	
Number Retrieved:						
Native:	10	264.01	5	267.12	5	259.68
FTP:	5	551.79	5	346.05	3	555.48
NFS:	17	235.28	5	235.51	5	245.39
RCP:	6	221.91		209.13	5	188.94
500M						
Number Archived:	4		4		2	
Number Retrieved:						
Native:	4	1052.84	4	1051.11	2	750.98
FTP:	4	2213.84	4	2205.46	2	2172.53
NFS:	4	916.88	2	925.86	2	908.52
RCP:	3	811.66	2	922.79	2	941.93
Total Host Files 5,890 (Limited by Secondary & Tertiary Storage Capacity)						

Table 5-3. Epoch Client - Data Distribution (1 of 2)

	Directory Level 1		Directory Level 2		Directory Level 3	
	files	avg time (sec)	files	avg time (sec)	files	avg time (sec)
64K						
Number Archived:	1000		1000		1000	
Number Retrieved:						
Native:	1000	1.03	900	1.05	600	1.11
FTP:	0	0	0	0	0	0
NFS:	1000	0.78	1000	1.03	1000	0.79
RCP:	0	0	0	0	0	0
100K						
Number Archived:	1000		1000		1000	
Number Retrieved:						
Native:	1000	1.50	1000	1.50	1000	1.46
FTP:	0	0	0	0	0	0
NFS:	1000	1.25	500	1.16	500	1.14
RCP:	0	0	0	0	0	0
500K						
Number Archived:	500		500		500	
Number Retrieved:						
Native:	500	3.69	200	4.09	50	4.59
FTP:	150	5.78	100	5.55	100	6.11
NFS:	500	3.59	200	3.36	50	3.37
RCP:	150	4.76	150	5.34	150	4.57
1M						
Number Archived:	50		20		20	
Number Retrieved:						
Native:	50	5.32	20	5.51	20	5.70
FTP:	25	8.78	10	9.44	10	9.37
NFS:	50	5.63	20	5.59	20	5.72
RCP:	25	6.58	10	7.01	10	7.01

Table 5-3. Epoch Client - Data Distribution (2 of 2)

	Directory Level 1		Directory Level 2		Directory Level 3	
	files	avg time (sec)	files	avg time (sec)	files	avg time (sec)
50M						
Number Archived:	5		5		5	
Number Retrieved:						
Native:	10	208.67	5	230.52	5	216.01
FTP:	5	346.05	3	351.39	3	357.96
NFS:	10	211.49	5	227.76	5	217.21
RCP:	5	211.16	3	218.723	3	217.50
Total Client Files 7,625 (Limited by Secondary & Tertiary Storage Capacity)						

5.6 LSR Testing

The data collected for the LSR portion was collected under a simulated system load of five (5) active requests. The LSR host (SGI Challenge) would send five requests to each FSMS connected (the Epoch Host and the AMASS servers) and then wait for a period of time to let the requests complete. Once the wait time expired, another batch of requests was sent. The files were transferred from the FSMS machines to the LSR host via FTP. A separate polling process ran every 5 seconds and checked a well known directory (/tmp/tool) for files which had names that matched a system UID. When a file was found, it was logged and then deleted. To get the times for retrieval, the logs had to be edited and the requests matched up with the responses. A tool for doing this was not written due to the small volume of data used.

This page intentionally left blank.

6. Conclusions

6.1 Network Limitations

Before stating the conclusions of this prototype, something must be said about the network environment. The Prototype 2 equipment used the same Local Area Network (LAN) segment 31 as the ECS servers and other equipment at Landover. Unlike Prototype 1 which relied on Federal Information Processing Standard (FIPS) for data transfers, Prototype 2 utilized Segment 31 for both control and data traffic. This caused significant performance degradation during specific periods on most days.

It was not possible to quantify the impact of this non-prototype LAN traffic on individual data samples. In an effort to reduce or eliminate the impact of this extraneous traffic, a larger number of data samples were collected and average transfer rates were used.

6.2 Prototype Approach

Several methods of managing a heterogeneous storage space were described in the Multi-FSMS White Paper. Most of these were deemed inappropriate for large search spaces due to the following:

1. Routing each request to each FSMS became impractical as the number of requests grew large. Only one FSMS held the requested file but each FSMS product searched for it. All but one of these searches ended in failure. In addition, while searching for a file that did not exist, the various FSMS products were not be able to service "legitimate" requests.
2. A biased search path and/or partitioning FSMS products by data set, was also inefficient as the search space grew large. Ingested data was stored to maximize retrieval efficiency based on anticipated access patterns and retrieval time constraints. This resulted in portions of the data sets overlapping multiple FSMS products and storage devices. In addition, some access patterns resulted in data file and/or data set movement from one data server to another. Thus, over time, a partitioning scheme would no longer represent the actual distribution of the data. Likewise, a very complex biasing scheme would be required to manage each data server.
3. Remote mounting all FSMS products via a NFS remained as a viable implementation method. This method did, however, suffer from inefficiencies due to server overhead particularly as the number of users grew large. There were also physical scaleability concerns. Both of these issues will be examined in greater detail in future prototypes.

The LSR approach was selected for this prototype. This method appeared to offer the most flexibility and fault tolerance. The LSR was implemented as a replicatable software module designed to identify file locations and route requests to the appropriate Data Server. In this case, the AMASS product resided on Data Server One and the EpochServ product resided on Data

Server Two. The file location was determined using a UID. The UID for each file consisted of the following:

Data Server ID	**D1** = AMASS, **D2** = EpochServ (For this Prototype)
FSMS ID (A data Server can consist of multiple FSMS products)	**AM** = AMASS, **ES** = EpochServ
Directory	**XXX** - Three Characters Mapped to a Table
File Number	**12345** - Five Numerals
File Extension	**XXX** - Three Characters Denoting File Type

The UID concept is used at several Distributed Active Archive Centers (DAACs) and was initially a concept targeted for ECS. UIDs will probably not be used in the future Version 1 ECS due to their limiting nature. As files migrate between data servers based on access profiles, UIDs will become inefficient. A database of file locations or some form of global name space manager will present a more scaleable and evolvable solution. Neither of these methods were specifically tested in Prototype 2 due to time constraints. They will be tested in future prototypes.

6.3 LSR Results

The LSR received file requests, translated them to the appropriate FSMS format, and routed the request to the appropriate data server with a return destination specified for the requested file. At this point, the individual FSMS product became responsible for handling the request. The LSR did not require specific acknowledgments from either the data server or the requester. Thus, a failure of the LSR would not result in the loss of previously transmitted requests. Multiple LSR modules could run simultaneously allowing failover.

Each series of requests was performed on an ordered set of data to minimize seek times between files. The requests for each directory level within each file size group were performed in a serial manner with little or no latency time. The host computers were not modified or rebooted for the duration of these tests.

In general, the LSR module worked very well and added a negligible amount of overhead to the overall request time. Unfortunately, the prototype suffered from implementation problems such as: 1) the disk space and other equipment constraints limited the number of files and file sizes stored and retrieved and 2) the Distributed Computing Environment (DCE) was not used for this prototype which resulted in a socket-based test tool design rather than a design using DCE-based Remote Procedure Calls (RPCs).

The socket-based implementation (described in Section 5) required spawning multiple child processes via Unix. Both host systems initially developed problems as the file size and hence the duration of each request increased and as the time between requests was reduced. This problem was corrected, for the most part, on the EpochServ Host (SunOS 4.1.3_U1) by modifying the method of managing child process communications. Unfortunately, the test team was unable to correct this problem on the AMASS Host (HP-UX A.09.04).

The process spawn problem on the HP manifested itself in two ways: 1) in some cases, a child process could not be spawned and resulted in no data point for a particular request and 2) in other cases, a request would complete, but some spawned processes would be suspended until additional requests entered the system. This is most evident in the 50 MB file retrievals.

An additional problem discovered during a discussion with AMASS Technical Support Personnel was that many of the retrievals graphed (Figures 6-1 through 6-9), appear to be from the disk cache as opposed to the tape library. The prototype team was unable to examine the contents of the disk cache directly because no adequate method of determining disk cache residency was available during testing. Utilities to examine and clear the disk cache will be made available in future releases of AMASS. In general, file retrievals under 10 seconds probably came from the disk cache.

Because of these problems, it was difficult to sustain data rates on both hosts. By modifying the time between requests, and later, by making some changes to the request spawn processing, the Epoch host provided a relatively stable performance curve. The HP host resisted all attempts to "normalize" the performance curve. Since neither the HP nor the Sun represent the target hardware for ECS, it was deemed inefficient to spend an extravagant amount of time attempting to correct the HP-UX problems.

Request handling for 5 requests per minute initially worked well until the OS anomalies manifested themselves. The problems with the AMASS performance seemed entirely dependent on the host computer and OS. There did not appear to be flaws in the product design. This was illustrated by the AMASS storage and retrieval data using native, FTP, NFS, and RCP methods (AMASS and Epoch data is included in Appendix A.). The following graphs display the request handling tests using a 5 requests per minute sampling. The lessons learned from the process spawn problem were incorporated into the ECS Design.

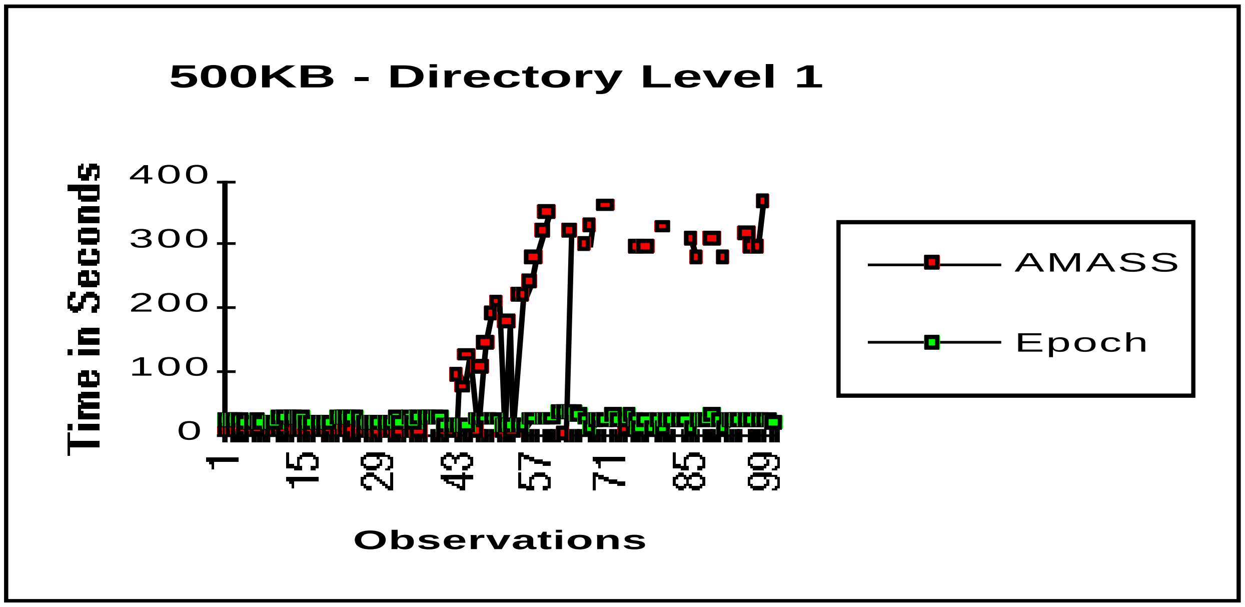

Figure 6-1. LSR 500 KB File Retrievals - Level 1

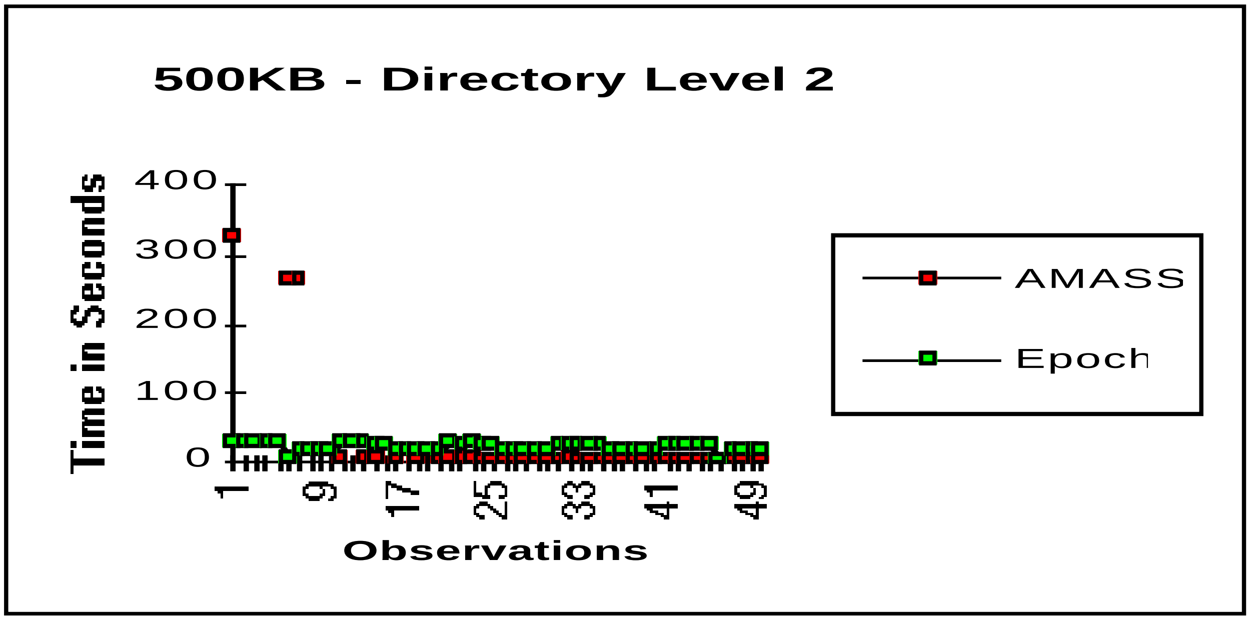

Figure 6-2. LSR 500 KB File Retrievals - Level 2

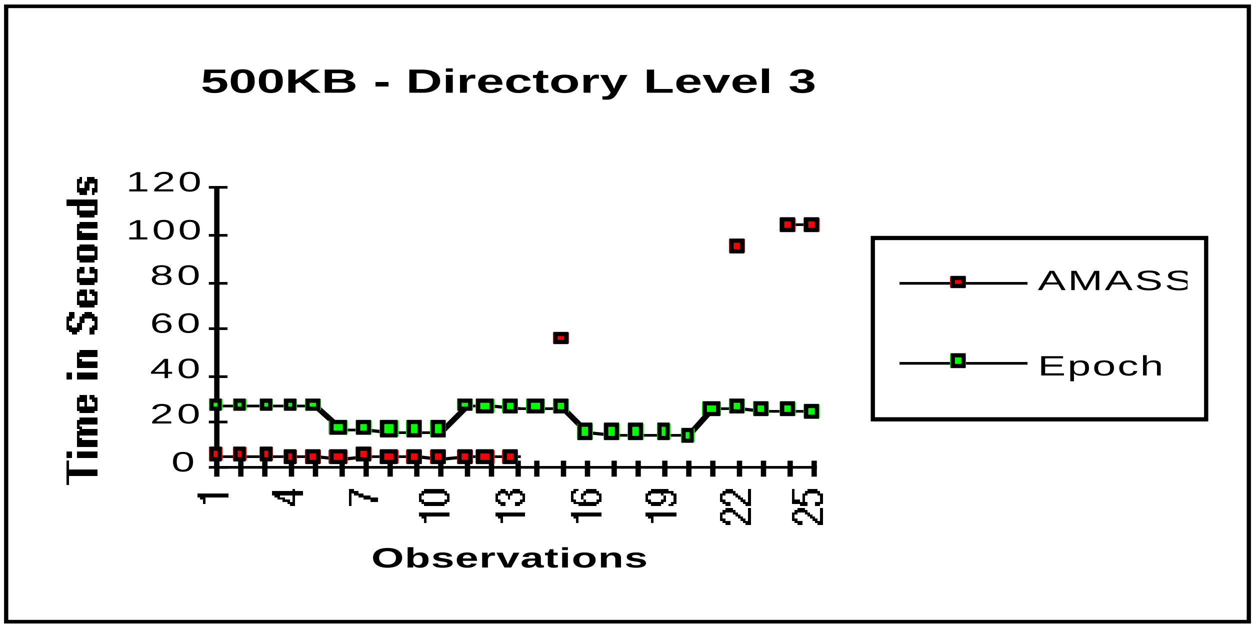

Figure 6-3. LSR 500 KB File Retrievals - Level 3

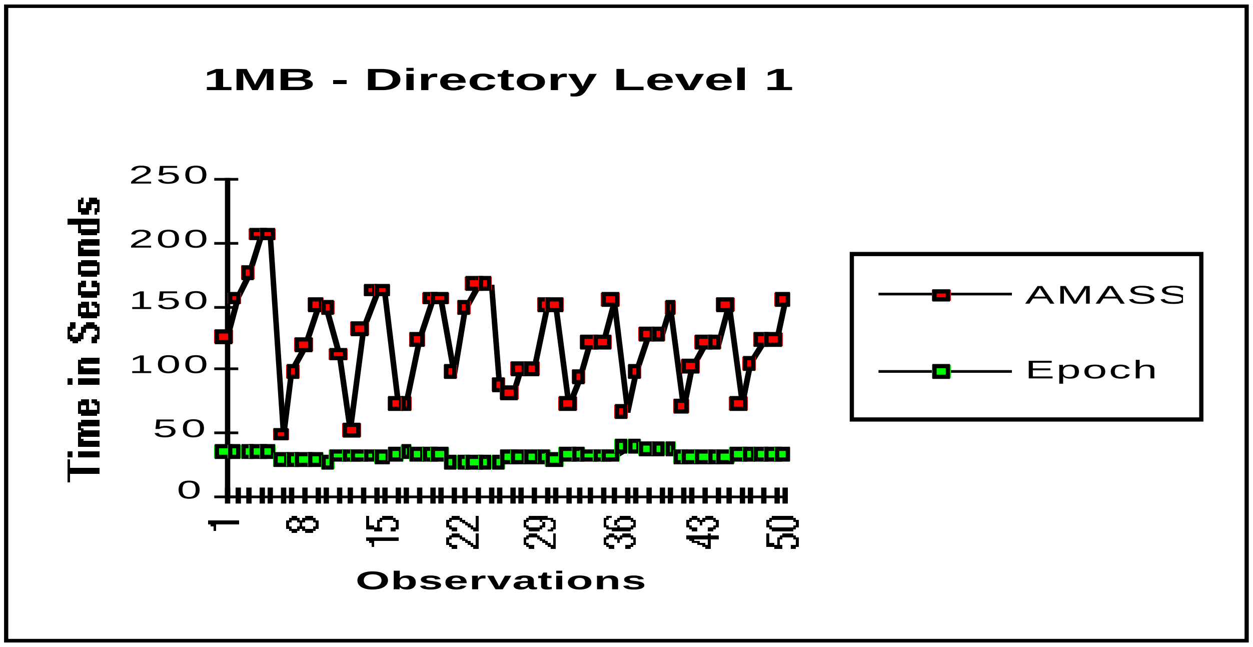

Figure 6-4. LSR 1 MB File Retrievals - Level 1

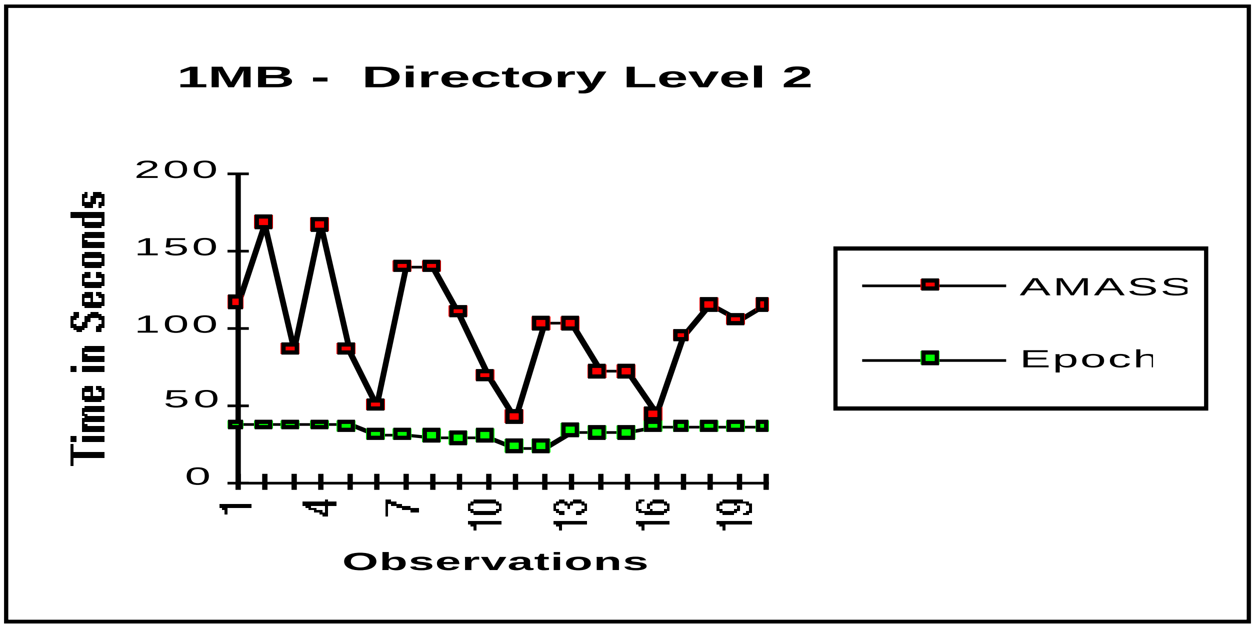

Figure 6-5. LSR 1 MB File Retrievals - Level 2

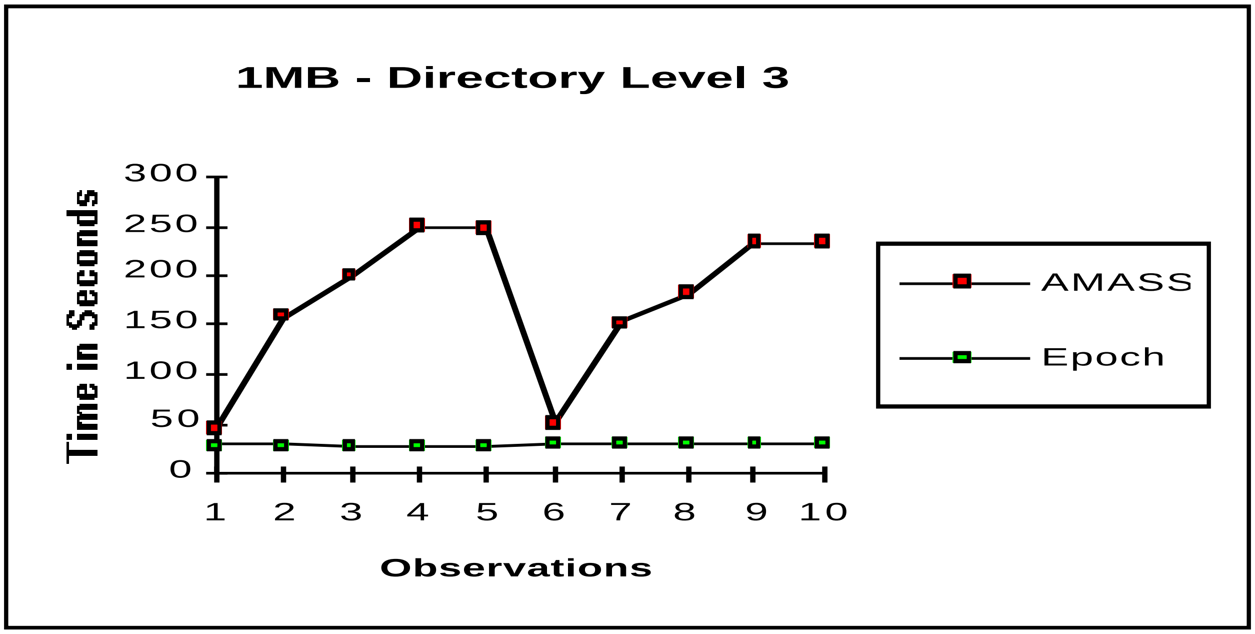

Figure 6-6. LSR 1 MB File Retrievals - Level 3

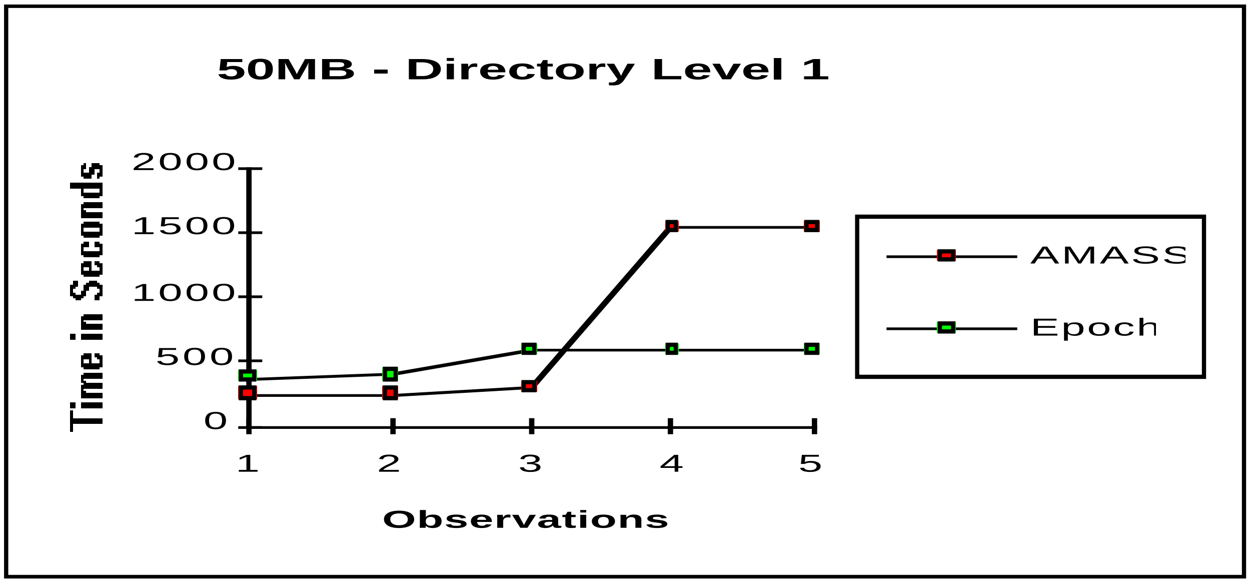

Figure 6-7. LSR 50 MB File Retrievals - Level 1

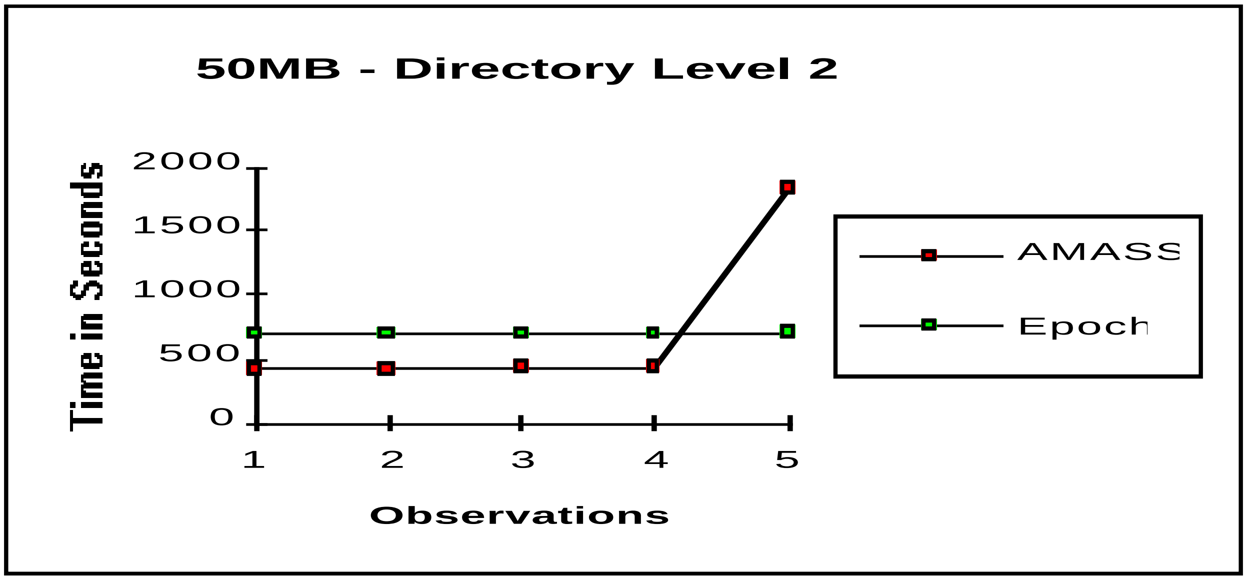

Figure 6-8. LSR 50 MB File Retrievals - Level 2

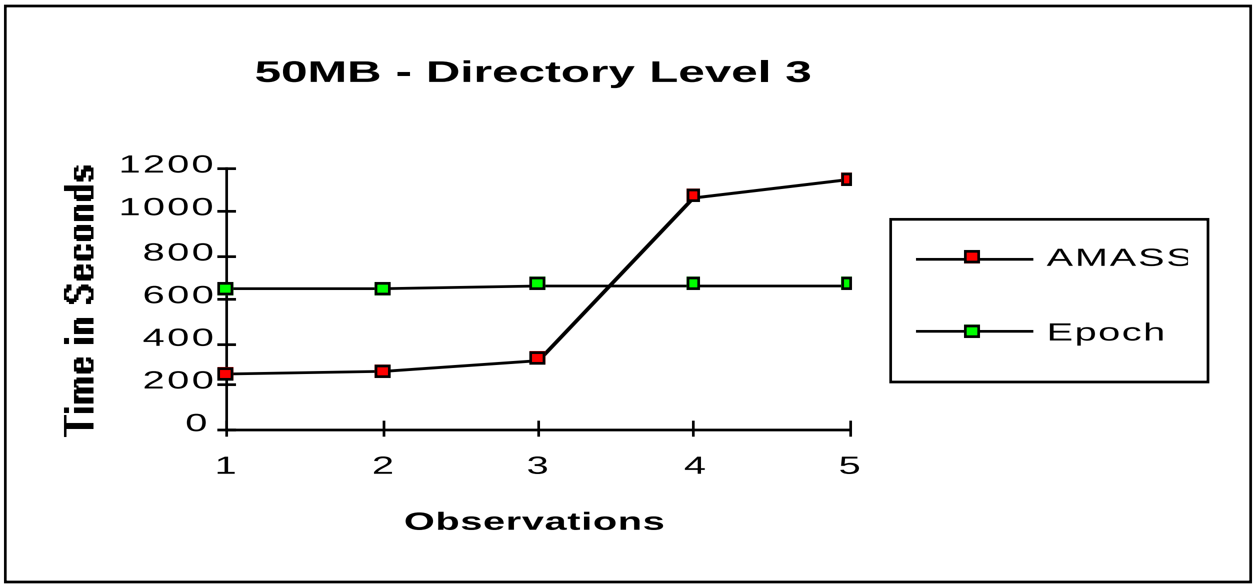

Figure 6-9. LSR 50 MB File Retrievals - Level 3

7. Problem & Resolution Summary

Sun / Epoch / HP Optical Jukebox:

- Coordinated the dispatch of a SUN Service Technician to repair the external 1.3GB disk drive on the SPARCstation2 (kolobok). The disk was set up as a single partition named / FSMS.
- Diagnosed an optical disk drive failure (Drive 2) on the HP Optical Jukebox. Coordinated the dispatch of a HP Service Technician to repair the second optical disk drive in the HP Jukebox.
- Revised the Epoch staging watermarks on both the Epoch server (kolobok) and client (babka) to remedy a problem with incoming file staging processes locking up due to the inability of Epoch to prestage enough files to create adequate space in the target file system.

HP 9000/800 - E30 / AMASS / Metrum Jukebox:

- Reallocated disk space on the HP 9000/800 to resolve a problem with a shortage of space in the /usr file system.
- Applied two software patches to the AMASS software; one to fix the change authorization script and the other to fix the AMASS manual startup process.

7.1 Epoch Problems

Week of 23 May 94

Epoch Client

While archiving a 400 MB file via the *epstage* command, the user received a "/file system full" error message.

This occurred during a operational client to host store operation, not create operation. The host should not have been full. The client was not full. Is this a watermarking problem? During *epstage*, an existing file should be transferred to the host for storage. Therefore, the file system should not be full. The software did not recover from this error.

Resolution:

The watermarks were originally set very high to facilitate immediate migration from the client to the host's storage. This caused a secondary problem of not having enough space in the file system to create a candidate list for migration, prior to filling the file system. The water marks were lowered to alleviate the problem.

Epoch Host

While performing an `epbsi` on a group of files, the disk file system became full. 236 MB of a 400 MB file filled the remaining free space in the file system. No error messages were received and the Epoch product failed to recover and it did not notify the user of a problem. A "cntl-c" was performed to terminate the command. A message to the effect of "Could not do EPBSI because operation would block" was received.

Resolution:

This problem was repeatable but no attempt was made to free space while it was waiting. At Epoch's suggestion, we lowered the watermarks to allow more time (and space in the file system) for Epoch to create candidate lists prior to filling the file system. This did seem to correct this problem.

7.2 AMASS Problems

Week of 7 July 1994

July 7, 1994:

Mr. Bruce Clark, of Electronics Data Systems (EDS), requested that Arnold Felix, of R-Squared (R^2), assist with the diagnosis of the read errors received when trying to retrieve files from tapes (primarily volume 3) loaded in the Metrum RSS-48 jukebox. The following section outlines the steps taken to investigate and replicate the errors:

1. The file system database record ids were verified as valid for the files and directories that had been experiencing read errors using the AMASS "fileprint" utility.
2. The files from a tape volume 26 that had not experienced errors were successfully retrieved.
3. The tape volume 3 was returned to an active and on-line state using the AMASS "volstat" and "volloc" commands.
4. The files from the tape volume 3 that had been experiencing errors were successfully retrieved. Retrievals were successful from volume 3 into the /tmp/r2 (created for this test), /users/bclark, and /fsms2/staging directories.
5. The condition of a power failure during the execution of a Unix "cp" command to copy files from the /archive/f3_100k/l2 directory (volume 3) was recreated. After the copy began successfully, the power to the tape drive was interrupted by momentarily turning off external power to both tape drives. The "cp" command hung suspended until terminated by issuing a "cntl-c" from the console keyboard.

6. At that time, the system console displayed the following series of Library Input/Output (LIBIO) daemon error messages consistent with the error messages we had received during previous DADS testing runs below.

```
Jul  5 15:44:59 repka LIBIO1_2[537]: <AMASS_S_0135> CDB that failed= 0x2b 0x0 0x0 0x0 0x0 0x3 0x11 0x0 0x0 0x0 0x0

Jul  5 15:44:59 repka LIBIO1_2[537]: <AMASS_S_0135> CDB that failed= 0x2b 0x0 0x0 0x0 0x0 0x3 0x11 0x0 0x0 0x0 0x0

Jul  5 15:44:59 repka LIBIO1_2[537]: <AMASS_S_0135> CDB status = 0x400

Jul  5 15:44:59 repka LIBIO1_2[537]: <AMASS_S_0135> CDB status = 0x400

Jul  5 15:44:59 repka LIBIO1_2[537]: <AMASS_S_0135> CDB data xfer = 0x0

Jul  5 15:44:59 repka LIBIO1_2[537]: <AMASS_S_0135> CDB data xfer = 0x0

Jul  5 15:44:59 repka LIBIO1_2[537]: <AMASS_S_0138> cdb_status = 0x400

Jul  5 15:44:59 repka LIBIO1_2[537]: <AMASS_S_0138> cdb_status = 0x400

Jul  5 15:44:59 repka LIBIO1_2[537]: <AMASS_W_0012> Read 0 on volume 3 on drive 2 (jukebox 1) failed

Jul  5 15:44:59 repka LIBIO1_2[537]: <AMASS_W_0012> Read 0 on volume 3 on drive 2 (jukebox 1) failed
```

7. The error messages were replicated a second time using the same power failure simulation procedures.
8. At this point, it appeared that the source of the drive and jukebox errors, that had been received during previous DADS testing runs, were associated with the momentary interruptions of power to the Metrum RSS-48 hardware during the frequent thunderstorms we had been experiencing.
9. All of the DADS String 2 hardware was connected to a circuit protected by an Uninterruptible Power Supply (UPS) in an effort to remove the momentary power interruptions as a potential source of the problem. The proposed course of action was to continue to monitor the condition of the hardware during power outages using the messages issued to the /usr/adm/syslog during interruptions of Alternate Current (AC) power by the UPS connected to the HP9000/800 server. This action cleared up the problem.

Week of 14 July 1994

July 14, 1994:

The AMASS software experienced additional system errors that disabled both drives in the Metrum RSS-48 jukebox. The following section outlines the steps taken to investigate and replicate the errors:

1. The /usr/adm/syslog files were examined to determine what error had caused AMASS to stop.
2. At 0300, the system had attempted to perform a nightly backup of the AMASS index files. At that time, AMASS failed with a series of Command Descriptor Block (CDB) errors and was unable to return the backup volume to it’s home slot. The backup failed, drive 2 was inactivated, and the AMASS system hung.
3. That afternoon, an attempt was made to restore AMASS to a functional state by stopping and starting AMASS from the Unix command line. At that time, AMASS could not be inactivated by the “amassstat” command because existing “cp” processes were attempting to read files from the AMASS system. The inactivation of AMASS 'timed out' waiting for the hung “cp” processes to complete.
4. Having failed to inactivate the AMASS system from the Unix command line, the server was rebooted. The jukebox robot reinitialized, but failed to eject and return the volumes in the drives to their respective slots in the tape drum.
5. While the server rebooted, a warning message that drive 2 was inactive (as expected) was posted to the system console. The drive was reactivated and an attempt was made to copy file D1AMbaa03423.DAT from volume 2 to /tmp. At that time, the system hung again. It issued a message to the syslog file that both drives were full, and the driveswere being marked out of service. Troubleshooting the system was suspended until the AMASS technical support staff could be contacted to discuss the problems.

July 15, 1994, A.M.:

1. In the morning, both drives were in an inactive state, and the “cp” process that had been started the previous evening was still hung waiting to complete. Upon attempting to reinitialize drive 1, the robot arm moved, found the drive was full, and inactivated itself again. An unsuccessful attempt was made to kill the hung “cp” process so the parent “csh” process was killed orphaning the “cp” process. Drive 2 was then reinitialized and the robot attempted to load volume 2, and it failed with a 'drive full' error message.
2. Mr. Felix, the local AMASS technical support representative, was then contacted. He expressed surprise that the drives had not ejected the loaded volumes upon rebooting the system. He recommended that the server be shutdown, the Small Computer System Interface (SCSI) connections between the server and the Metrum jukebox be verified, and the server restarted. A site visit was also set up for the afternoon.

3. An attempt to inactivate the AMASS system from the Unix command line was once again unsuccessful because the AMASS system was unable to unmount the /archive mount point (device busy), kill the AMASS daemons (because the file system was still mounted), or inactivate AMASS using “amassstat” (because the system still had file requests open).

4. The server was then halted, the connections on the SCSI bus and the Parallan Single-ended / Differential converter checked, power to the Parallan converter cycled, and the system was restarted.

5. When the system restarted, AMASS started, the jukebox robot ejected the volumes from the drives and replaced them in their slots, and the system came back up with all previous “cp” processes terminated. Both drives were still inactivated, so both drives were successfully reinitialized, without the system issuing any error messages.

6. The file that had caused the error the day before (D1AMbaa03423.DAT) was copied from the AMASS system to /tmp successfully.

July 15, 1994, P.M.:

1. In the afternoon, Mr. Felix arrived to assist with identifying the cause of the earlier system failures. By this time, the AMASS system had been up and running and testing had been able to resume without additional failures.

2. The syslog error messages and the steps taken to restore AMASS to a functional state were reviewed with Mr. Felix. Copies of the syslog files captured for further research were forwarded to the R2 and Advanced Archival Products (AAP) technical support staff. No definite conclusions regarding the failure of the system on the morning of July 14 were reached.

3. In the course of checking the sanity of the AMASS system, the AMASS database check utilities, “dbcheck” and “sysdbchk”, were run. At that time, the diagnostics detected a corrupted file on volume 12 (a volume on which we had seen previous failures). The “sysdbchk” error message indicated that the file /archive/f3_100m/D1AMlaa38645.DAT had been written past the end of the tape on volume 12.

4. An attempt was made to copy the corrupted file to /tmp and the LIBIO error condition that had been observed in previous syslog (when attempting to read from volume 12) was successfully duplicated. The syslog error was:

```
Jul 15 17:03:53 repka LIBIO1_2[2256]: <AMASS_W_0012> Read 0 on volume 12 on drive 2
(jukebox 1) failed

Jul 15 17:03:53 repka LIBIO1_2[2256]: <AMASS_W_0012> Read 0 on volume 12 on drive 2
(jukebox 1) failed

Jul 15 17:03:53 repka LIBIO1_2[2256]: <AMASS_E_0118>  SCSI Sense information dump (in
hex):
```

```
Jul 15 17:03:53 repka LIBIO1_2[2256]: <AMASS_E_0118>  SCSI Sense information dump (in
hex):

Jul 15 17:03:53 repka LIBIO1_2[2256]: <AMASS_E_0118>  Sense key: 0x0  ASC: 0x0 ASCQ:
0x0

Jul 15 17:03:53 repka LIBIO1_2[2256]: <AMASS_E_0118>  Sense key: 0x0  ASC: 0x0 ASCQ:
0x0

Jul 15 17:03:53 repka LIBIO1_2[2256]: <AMASS_E_0118>   00 00 00 00 00 00 00 00

Jul 15 17:03:53 repka LIBIO1_2[2256]: <AMASS_E_0118>   00 00 00 00 00 00 00 00
```

5. It was agreed upon to delete the file in order to prevent this condition from generating further errors. There was no definite conclusion as to why the file was corrupted, however, Mr. Felix speculated that a failed write during a power failure could have caused the end of file to be corrupted.
6. Once the corrupted file had been deleted, “dbcheck” and “sysdbchk” were successfully run against the AMASS file system database.

7.3 AMASS Power Fault Testing

July 26, 1994:

1. A process to move (mv) file /fsms2/staging/D1AMraa46843.DAT to /archive/f2_5g was started.
2. Power to drive 1 was interrupted by pulling the power plug on the back of the drive unit.
3. AMASS issued the following error messages to the /usr/adm/syslog:

```
Jul 26 14:12:59 repka LIBIO1_1[482]: <AMASS_S_0139> SCTL_INCOMPLETE

Jul 26 14:12:59 repka LIBIO1_1[482]: <AMASS_S_0139> SCTL_INCOMPLETE

Jul 26 14:12:59 repka LIBIO1_1[482]: <AMASS_S_0135> CDB that failed= 0x34 0x0 0x0 0x0
0x0 0x0 0x0 0x0 0x0 0x0 0x0

Jul 26 14:12:59 repka LIBIO1_1[482]: <AMASS_S_0135> CDB that failed= 0x34 0x0 0x0 0x0
0x0 0x0 0x0 0x0 0x0 0x0 0x0

Jul 26 14:12:59 repka LIBIO1_1[482]: <AMASS_S_0135> CDB status = 0x400

Jul 26 14:12:59 repka LIBIO1_1[482]: <AMASS_S_0135> CDB status = 0x400

Jul 26 14:12:59 repka LIBIO1_1[482]: <AMASS_S_0135> CDB data xfer = 0x0

Jul 26 14:12:59 repka LIBIO1_1[482]: <AMASS_S_0135> CDB data xfer = 0x0
Jul 26 14:12:59 repka LIBIO1_1[482]: <AMASS_S_0138> cdb_status = 0x400

Jul 26 14:12:59 repka LIBIO1_1[482]: <AMASS_S_0138> cdb_status = 0x400

Jul 26 14:12:59 repka LIBIO1_1[482]: <AMASS_S_0135> CDB that failed= 0x1 0x0 0x0 0x0
0x0 0x0 0x0

Jul 26 14:12:59 repka LIBIO1_1[482]: <AMASS_S_0135> CDB that failed= 0x1 0x0 0x0 0x0
0x0 0x0 0x0

Jul 26 14:12:59 repka LIBIO1_1[482]: <AMASS_S_0135> CDB status = 0x400

Jul 26 14:12:59 repka LIBIO1_1[482]: <AMASS_S_0135> CDB status = 0x400

Jul 26 14:12:59 repka LIBIO1_1[482]: <AMASS_S_0135> CDB data xfer = 0x0

Jul 26 14:12:59 repka LIBIO1_1[482]: <AMASS_S_0135> CDB data xfer = 0x0

Jul 26 14:12:59 repka LIBIO1_1[482]: <AMASS_S_0138> cdb_status = 0x400

Jul 26 14:12:59 repka LIBIO1_1[482]: <AMASS_S_0138> cdb_status = 0x400

Jul 26 14:13:19 repka LIBSCHED1[481]: <AMASS_E_1404> Unable to load volume 25.  Drive
2 (jukebox 1) is full - marked out of service

Jul 26 14:13:19 repka LIBSCHED1[481]: <AMASS_E_1404> Unable to load volume 25.  Drive
2 (jukebox 1) is full - marked out of service
```

4. The AMASS software placed both drives in an inactive state and placed volume 24 in an inactive, read-only state.
5. Attempts to kill the “mv” process were unsuccessful, but parent process was successfully killed.

July 28, 1994:

The established recovery procedures were then tried to restore AMASS Operations.

1. Both drives were reinitialized successfully.
2. An attempt to reinitialize volume 24, using the “volstat” command, failed with the error message: “VOLSTAT operation failed, AMASS volume currently in use”.
3. An attempt was made to inactivate AMASS, but because the system thought it had a file operation still in progress, the “amassstat” command timed-out.
4. An attempt to unmount/archive failed because the system detected an operation attempting to write a file to /archive/f2_5g and issued a “device busy” message.
5. An attempt to kill the AMASS daemons with the “killdaemons” command failed because AMASS was still running.
6. The system was then rebooted.

7. Following the system reboot, volume 24 was successfully reinitialized and made writeable.
8. The AMASS system was stopped and the "dbcheck" and "sysdbchk" utilities ran, neither of which encountered any errors.
9. AMASS was restarted.
10. The “mv” operation was reran to move the file /fsms2/staging/D1AMraa46843.DAT to /archive/f2_5g.
11. AMASS placed the file on volume 25 and completed the “mv” operation successfully.
12. Both the AMASS utilities “fileprint” and “volfilelist” show the file on volume 25, with the correct permissions and file size.

In discussing the failure scenario with Mr. Felix, he indicated that the system behavior was consistent with their expectations in the event of a catastrophic drive failure. In that situation, the following steps should occur:

1. The drive should try to eject the tape for movement to another drive.
2. The drive would be unable to eject the tape (due to the power loss), and the volume move operation would fail.
3. Once the volume move fails, AMASS would be unable to determine whether one or both drives had failed, and the AMASS software would inactive both drives.
4. The failed volume would then be placed in an inactive, read-only status.

The primary concern with this failure scenario is the fact that a catastrophic failure of one drive could potentially disable an entire Automated Tape Library (ATL). According to Mr. Felix, in most cases the ejection of a tape volume would be initiated by a soft error (such as an Error Correction & Control (ECC) error), in which case the drive would still be able to eject the tape. The hard failure (power interruption during a write) has the potential for more serious impact on the entire ATL and AMASS software configuration.

Appendix A. Prototype Data Points

For brevity the data collected for Prototype 2 will be available as a separate document in the ECS Library and the World Wide Web.

This page intentionally left blank.

Abbreviations And Acronyms

AAP	Advanced Archival Products
AC	Alternating Current
AMASS	Archival Management and Storage System
ATL	Automated Tape Library
CDB	Command Descriptor Block
cp	copy
CPU	Central Processing Unit
DAAC	Distributed Active Archive Center
DADS	Data Archive and Distribution System
DCE	Distributed Computing Environment
DID	Data Item Description
DR	Disaster Recovery
ECC	Error Correction & Control
ECS	EOSDIS Core System
EOSDIS	Earth Observing System Data and Information System
EPBSI	Epoch Bulk Stage In
FIPS	Federal Information Processing Standard
FSMS	File Storage Management Systems
FTP	File Transfer Protocol
GB	Giga Byte
HP	Hewlett Packard
HWM	High Water Mark
Inode	Index Node
KB	Kilo Byte
LAN	Local Area Network
LIBIO	Library Input/Output
LSR	Location Server/Request Router
LWM	Low Water Mark

MB	Mega Byte
MFD	Master File Directory
MLS	Multi-Level Staging
MSRM	Mass Storage Reference Model
mv	move
NFS	Network File System
OS	Operating System
PSWM	Pre-Stage Water Mark
R/W	Read/Write
R^2	R-Squared
RCP	Remote Copy
RPC	Remote Procedure Call
SCSI	Small Computer System Interface
SDPS	Science Data Processing Segment
SGI	Silicin Graphics, Inc.
SPARC	Single Processor Architecture
SUN	Sun MicroSystems
SVHS	Super Vertical Helical Scan
UFS	UNIX File System
UID	Universal Identifier
UPS	Uninteruptible Power Supply
VFS	Virtual File System

Bibliography

Figure 3-1 with some modifications, is taken from: Bach, Maurice, J., The Design of the UNIX Operating System, Prentice-Hall, 1986, pg. 20.

Stevens, W. Richard, UNIX Network Programming, Prentice-Hall, 1990.

AMASS^{TM}Archival Management and Storage System - Reference Manaual Version 4.2, Advanced Archival Procuts Inc., April 15, 1994.

EpochServ System Administrator's Guide - Release 6.0, Epoch Systems Inc., August 1993.

EpochMigration SystemAdministrator's Guide, Epoch Systems Inc., March 1993.